The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Parallel relaxed and extrapolated algorithms for computing PageRank

Josep Arnal · Héctor Migallón · Violeta Migallón · Juan A. Palomino · José Penadés

Received: date / Accepted: date

Abstract In this paper parallel Relaxed and Extrapolated algorithms based on the Power method for accelerating the PageRank computation are presented. Different parallel implementations of the Power method and the proposed variants are analyzed using different data distribution strategies. The reported experiments show the behavior and effectiveness of the designed algorithms for realistic test data using either OpenMP, MPI or an hybrid OpenMP/MPI approach in order to exploit the benefits of shared memory inside the nodes of current SMP supercomputers.

Keywords PageRank · Parallel algorithms · Power method · Relaxation and extrapolation

1 Introduction

The PageRank algorithm for determining the importance of Web pages has become a central technique in Web search. PageRank is essentially the stationary distribution vector of a Markov chain whose transition matrix is a convex combination of the Web link graph and a certain rank 1 matrix. Due to the large size and sparsity of the matrix, methods based on decomposition are considered infeasible; instead, iterative methods are used, where the

This research was partially supported by the Spanish Ministry of Science and Innovation under grant number TIN2011-26254.

J. Arnal · V. Migallón · J. A. Palomino · J. Penadés
Departamento de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante, 03071 Alicante, Spain
E-mail: violeta@dccia.ua.es

H. Migallón
Departamento de Física y Arquitectura de Computadores, Universidad Miguel Hernández, 03202 Elche, Alicante, Spain

computation is dominated by matrix-vector products; see e.g., [1]. Traditionally, PageRank has been computed using the Power method. Several methods to accelerate the Power method have been developed such as extrapolation methods, block-structure methods or adaptive methods; see e.g., [7], [8], [9], [16] and the references cited therein. In recent years opportunities for parallel execution have broadened their scope. By using the linear system formulation of the PageRank problem, in [5] parallel algorithms for computing PageRank based on Krylov subspace iterative methods are analyzed. In [4] an inner-outer algorithm is investigated. The proposed technique is a preconditioning approach based on the observation that the smaller the damping factor is, the easier it is to solve the problem. This inner-outer method compares favorably with other widely used schemes. Moreover its parallel implementation achieves a substantial gain with respect to the Power method especially when the damping factor is close to 1; other related works can be found e.g., in [10], [13] and [14].

In this paper parallel Relaxed and Extrapolated algorithms based on the Power method that accelerate its convergence are presented. The remainder of the paper is structured as follows. In Section 2 we provide a brief description of the PageRank problem and we introduce the Power method and some acceleration methods based on the extrapolation technique. In Section 3, parallel algorithms based on the Power method using relaxation and/or extrapolation are introduced. The numerical experiments performed in Section 4 show the behavior of these algorithms using different data distribution strategies on both shared and distributed memory multicore architectures. Finally, in Section 5 we give some conclusions.

2 Computing the PageRank

PageRank [12] is a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. The PageRank problem can be seen as a matrix problem. Let $G = [g_{ij}]_{i,j=1}^{n}$ be a Web graph adjacency matrix with elements $g_{ij} = 1$ when there is a link from page j to page i, with $i \neq j$, and zero otherwise. Here n is the number of Web pages. From this matrix we can construct a transition matrix $P = [p_{ij}]_{i,j=1}^{n}$ as follows: $p_{ij} = \frac{g_{ij}}{c_j}$ if $c_j \neq 0$ and 0 otherwise, where $c_j = \sum_{i=1}^{n} g_{ij}$, $1 \leq j \leq n$, represents the number of out-links from a page j. For pages with a nonzero number of out-links, i.e., $c_j \neq 0$ for all j, $1 \leq j \leq n$, the matrix P is column stochastic. Thus each element of this matrix has values between 0 and 1, and the sum of the components of each column is 1. In this case the PageRank vector can be obtained by solving $Px = x$. Since we are interested in a probability distribution, the sum of the components of x is assumed to be one. Algorithm 1 shows the original Power method [15] for the PageRank computation where $e = (1, 1, \ldots, 1)^T$. Note that we use the L_1 norm $\|x\|_1 = \sum_{i=1}^{n} |x_i|$. When the matrix $P \geq 0$ is irreducible (i.e., its graph is strongly connected) and stochastic, its largest eigenvalue in magnitude is $\lambda_{max} = 1$.

Algorithm 1: Power method.

Initialization $x^0 = \frac{e}{n},\ k = 0;$
repeat
  $x^{k+1} = Px^k;$
  $\delta = \|x^{k+1} - x^k\|_1;$
  $k = k + 1;$
until $\delta < \epsilon;$

Thus, Algorithm 1 converges to the eigenvector corresponding to $\lambda_{max} = 1$, and when normalized, it is the stationary probability distribution over pages under a random walk on the Web. However, the Web contains many pages without out-links, called dangling nodes. Dangling pages present a problem for the mathematical PageRank formulation because in this case the matrix P is non-stochastic and then Algorithm 1 can not be used. Moreover, the matrix irreducibility is not satisfied for a Web graph. In order to overcome these difficulties, Page and Brin [12] change the transition matrix P to a column stochastic matrix $\bar{P} = \alpha(P + vd^T) + (1 - \alpha)ve^T$, where $d \in \Re^n$ is the dangling page indicator defined by $d_i = 1$ if and only if $c_i = 0$ and the vector $v \in \Re^n$ is some probability distribution over pages. This model means that the random surfer jumps from a dangling page according to a distribution v. For this reason v is called a teleportation distribution. Originally uniform teleportation $v = \frac{e}{n}$ was used. Consequently, v is also known as a personalization vector. Then, setting α such that $0 < \alpha < 1$ the matrix $\bar{P}$ is column stochastic, irreducible and it preserves the L_1 norm, that is, $\|\bar{P}x\|_1 = \|x\|_1$ and therefore we can reformulate Algorithm 1 using the matrix $\bar{P}$. That is, Algorithm 1 is utilized in order to solve the stationary distribution of the ergodic Markov chain defined by $\bar{P}$, $\bar{P}x = x$, obtaining the corresponding algorithm: Note that although the

Algorithm 2: Power method for solving $\bar{P}x = x$.

Initialization $x^0 = \frac{e}{n},\ k = 0;$
repeat
  $x^{k+1} = \alpha Px^k;$
  $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1;$
  $x^{k+1} = x^{k+1} + \gamma v;$
  $\delta = \|x^{k+1} - x^k\|_1;$
  $k = k + 1;$
until $\delta < \epsilon;$

matrix $\bar{P}$ is dense, Algorithm 2 has been designed such that it is not necessary to construct explicitly the matrix $\bar{P}$.

A key parameter in this model is the damping factor α that determines the weight given to the Web link graph in the model. In the original formulation of PageRank [12] the Power method was applied using $\alpha = 0.85$. However, a higher value of α (close to 1) yields a model that is mathematically closer

to the actual link structure of the Web but makes the computation more difficult [6]. This parameter α controls the asymptotic rate of convergence and as $\alpha \to 1$, the expected number of iterations required for convergence increases dramatically and new approaches for accelerating the PageRank computation are required. In fact, the calculation of many PageRank vectors with different values of α looks promising for the design of anti-spam mechanism [17].

The extrapolation algorithms [6] accelerate the convergence of PageRank by using successive iterates of the Power method to estimate the nonprincipal eigenvectors of the hyperlink matrix, and periodically subtracting these estimates from the current iterate of the Power method. In [9] a quadratic extrapolation algorithm, based on the same idea as Aitken extrapolation, was presented. This work assumed that none of the nonprincipal eigenvalues of the hyperlink matrix were known. The Extrapolation Power methods treated here exploit the knowledge of eigenvalues of the hyperlink matrix. Moreover, the extrapolation needs to be applied only once; see e.g., [6]. Note that if

Algorithm 3: Extrapolation Power method.

Initialization $x^0 = \frac{e}{n},\ k = 0;$
repeat
  $x^{k+1} = \alpha P x^k;$
  $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1;$
  $x^{k+1} = x^{k+1} + \gamma v;$
  if $k+1 == r+2$ **then** $x^{k+1} = \frac{x^{k+1} - \alpha^r x^{k+1-r}}{1-\alpha^r};$
  $\delta = \|x^{k+1} - x^k\|_1;$
  $k = k+1;$
until $\delta < \epsilon;$

$r = 1$ a simple extrapolation is used at iteration 3, while if $r = 2$ a quadratic extrapolation is applied once at iteration 4.

3 Parallel Algorithms

In order to design the parallel algorithms, we consider that P is partitioned into p row blocks. Each block P_i, $1 \leq i \leq p$, is a matrix of order $n_i \times n$, with $\sum_{i=1}^{p} n_i = n$. Analogously, we consider the vectors x^k and v partitioned according to the block structure of P. Obviously, the Power method for solving $\bar{P}x = x$ can be executed in parallel. In this case each process actualizes a block of the vector x^{k+1} and a synchronization of all processes is performed at each iteration in order to construct the global iterate vector x^{k+1}. Due to this synchronization, we can use the formulation of Algorithm 2 because the property of preserving the L_1 norm remains valid. Taking into account that the Power method is equivalent to use a Jacobi type splitting for solving the linear system $(I - \bar{P})x = 0$, in order to improve the rate of convergence we could use a relaxed Jacobi type splitting. Thus, a relaxation parameter

Algorithm 4: Parallel Power method.

Initialization $x^0 = \frac{e}{n},\ k = 0;$
repeat
  for $i = 1, 2, \ldots, p,$ **do in parallel**
    $x_i^{k+1} = \alpha P_i x^k;$
    $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1;$
    $x_i^{k+1} = x_i^{k+1} + \gamma v_i;$ (1)
  end
  $x^{k+1} = [x_1^{k+1}, \ldots, x_p^{k+1}];$
  $\delta = \|x^{k+1} - x^k\|_1;$
  $k = k + 1;$
until $\delta < \epsilon;$

$\beta > 0$ can be introduced and replace the computation of x_i^{k+1} in (1) with the equation $x_i^{k+1} = \beta(x_i^{k+1} + \gamma v_i) + (1 - \beta)x_i^k$. Clearly, with $\beta = 1$ equation (1) is recovered. In the case of $\beta \neq 1$ we have a parallel Relaxed Power method. Note that the relaxation parameter needs to be chosen in such a way that the convergence of the method is assured. In order to accelerate the convergence of Algorithm 4 we also propose parallel Relaxed Extrapolated algorithms based on Algorithm 3 as follows. In Algorithm 5 the relaxation is applied only after the extrapolation is performed. Note that the computation of $\|x^{k+1}\|_1$ and

Algorithm 5: Parallel Relaxed Extrapolated Power method.

Initialization $x^0 = \frac{e}{n},\ k = 0;$
repeat
  for $i = 1, 2, \ldots, p,$ **do in parallel**
    $x_i^{k+1} = \alpha P_i x^k;$
    $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1;$
    $x_i^{k+1} = x_i^{k+1} + \gamma v_i;$
    if $k + 1 == r + 2$ **then** $x_i^{k+1} = \frac{x_i^{k+1} - \alpha^r x_i^{k+1-r}}{1-\alpha^r};$
    if $k + 1 > r + 2$ **then** $x_i^{k+1} = \beta x_i^{k+1} + (1 - \beta)x_i^k;$ (2)
  end
  $x^{k+1} = [x_1^{k+1}, \ldots, x_p^{k+1}];$
  $\delta = \|x^{k+1} - x^k\|_1;$
  $k = k + 1;$
until $\delta < \epsilon;$

δ in Algorithms 4 and 5 is also performed in parallel in such a way that each process i computes the portions $\|x_i^{k+1}\|_1$ and $\|x_i^{k+1} - x_i^k\|_1$ followed by a reduction of these values. For the sake of simplicity, we have omitted these computations from the formulation of Algorithms 4 and 5. We would like to point out that if $\beta = 1$ in equation (2), Algorithm 5 reduces to the parallel counterpart of Algorithm 3.

4 Experimental setup and results

We have implemented the algorithms described here on an HPC cluster of 26 nodes HP Proliant SL390s G7 connected through a network of low-latency QDR Infiniband-based. Each node consists of two Intel XEON X5660 hexa-core at up to 2.8 GHz and 12MB cache per processor, with 48 GB of RAM. The operating system is CentOS Linux 5.6 for x86 64 bit. The parallel environment has been managed using both MPI (Message Passing Interface) [3] and OpenMP (Open Multi-Processing) [11]. That is, an hybrid MPI/OpenMP implementation has been designed by combining various OpenMP threads for each MPI process. That is, MPI is used for data distribution, and OpenMP in order to perform the computation inside the cores of each node. Concretely, let p be the number of processes performed, $p = s * c$ indicates that s nodes of the parallel platform have been used and for each one of these nodes, c OpenMP threads have been considered. Therefore, we use a philosophy of distributed shared memory using $p = s \times c$ processes or threads. Particularly, if $s = 1$ the algorithms are executed in shared memory by using $p = c$ threads on a single node. Conversely, if $c = 1$, we are working on distributed memory using $p = s$ nodes. In order to test the algorithms treated here we have used three datasets of different sizes, available from the Laboratory for Web Algorithmics (http://law.dsi.unimi.it). These transition matrices have been generated from a web-crawl [2]. Table 1 summarizes the main characteristics of the graphs

Graph	Nodes (n)	Arcs (nnz)	Dang. nodes	Density	Memory
it-2004	41,291,594	1,150,725,436	12.76%	27.87	4.75 GB
webbase-2001	118,142,155	1,019,903,190	23.41%	8.63	5.12 GB
uk-2007-05	105,896,555	3,738,733,648	12.23%	35.31	15.11 GB

Table 1 Graphs collection; n = number of nodes (matrix size), nnz = number of arcs (nonzero elements of the matrix), Dang. nodes=percentage of dangling nodes, Density=arcs/nodes, Memory = memory requirements using CSR' format.

used in this work. Note that, as the dimension of the link matrix grows, its relative sparseness increases as well. To compute PageRank for large domains there is no possible way to work with the matrix in its full format, the memory requirements would be too high. Therefore, a sparse matrix format is needed in order to store the matrices. Concretely, the Compressed Sparse Row (CSR) format was used, which is one of the most extensively used storage scheme for general sparse matrices, with minimal storage requirements. We represent the two vectors of indexes of the CSR format by integers without sign of 32 bits, while the values and the iterate vectors are represented by means of double precision floating point with 64 bits. The memory requirements (bits) needed to store a matrix with the original CSR format can be computed by the following expression $M_{CSR} = 32(n+1)+32nnz+64nnz \approx 32(n+3nnz)$ bits. Taking into account that, for each column of the matrix, all nonzero elements are equal to a fixed value, it is stored once in an ordered vector. In this way the memory

requirements to store the matrix are $M_{CSR'} = 32(n+1) + 32nnz + 64n \approx 32(3n + nnz)$ bits. For example, for the uk-2007-05 matrix with $105,896,555$ rows and $3,738,733,648$ nonzero elements, the original CSR format requires 42 GB of memory while the modification considered here requires 15.11 GB. Generally, this modified CSR format (CSR') [10] has involved a reduction of memory requirements of about $63 - 73\%$.

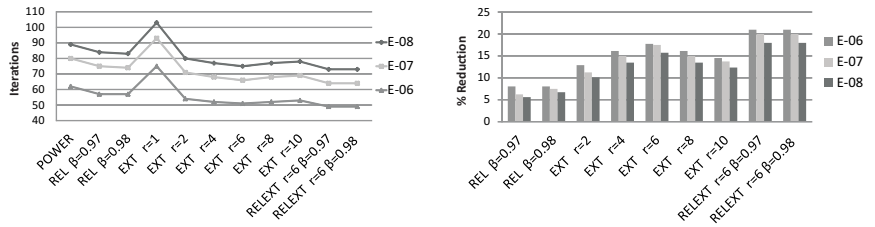
Implementing the PageRank calculations in a parallel environment opens several possibilities of data partitioning (i.e., how the data are divided among nodes) and load balancing (i.e., to ensure that all nodes perform similar amount of work). The most expensive operation performed in the calculation of the PageRank values is a matrix-vector multiplication. This is a perfectly parallel operation with several possible methods for partitioning both the matrix and the vector. We have considered two different methods for partitioning the link matrix among nodes. The first method we have chosen is a row-wise distribution (row-wise partitioning) where each node gets the same amount of rows. However, the number of nonzero elements per row of the link matrix used to calculate PageRank can differ immensely. In order to balance the calculations we consider a second matrix distribution strategy where each node has to handle the same amount of nonzero elements (nonzero elements partitioning). Algorithm 6 describes these distribution strategies, where $\omega_n, \omega_{nnz} \in \Re$ such that $\omega_n + \omega_{nnz} = 1$ attach a weight to the number of rows and to the number of nonzero elements, respectively. In this algorithm, n represents the number

Algorithm 6: Partitioning method.

```
umbral = (nω_n + nnz ω_nnz)/s;
PART_0 = cont = j = 0;
for i = 1, 2, ..., s − 1, do
    while cont ≤ umbral do
        j = j + 1;
        cont = cont + (ω_n + nnz_j ω_nnz);
    end
    j = j − 1;
    PART_i = j;
    cont = 0;
end
PART_s = n;
```

of rows of the matrix, nnz_j is the number of nonzero elements in the j-th row with $nnz = \sum_{j=1}^{n} nnz_j$, s is the number of partitions to be performing and the vector $[PART_0, \ldots, PART_s]$ stores the indexes of the partitions. When $\omega_n = 1$ and $\omega_{nnz} = 0$, a row-wise distribution is considered, while when $\omega_n = 0$ and $\omega_{nnz} = 1$, Algorithm 6 performs a nonzero elements partitioning.

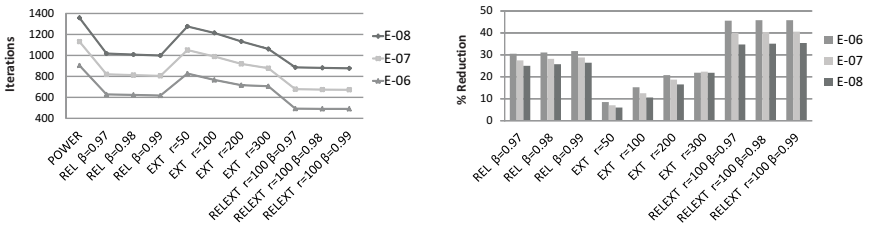
Of the algorithms we have discussed here for accelerating the convergence of PageRank, the combining of relaxation and extrapolation performs the best empirically. Figures 1 and 2 compare the convergence rates for the Power method and the Relaxed (REL) and/or Extrapolated (EXT) methods set-



(a) Number of iterations.

(b) Reduction with respect to the Power method (%).

Fig. 1 Comparison of convergence rates for computing PageRank, $\alpha = 0.85$, webbase-2001.



(a) Number of iterations.

(b) Reduction with respect to the Power method (%).

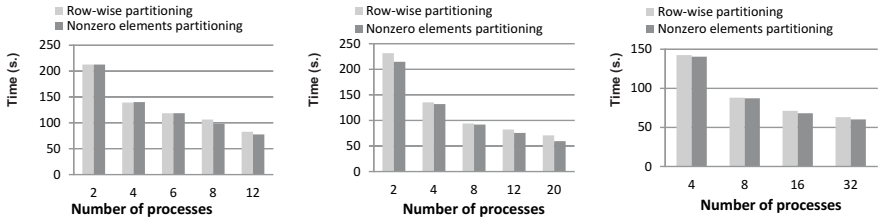
Fig. 2 Comparison of convergence rates for computing PageRank, $\alpha = 0.99$, webbase-2001.

ting a global convergence scheme and varying the stopping criterion for the webbase-2001 matrix. When $\alpha = 0.85$, the proposed Relaxed Extrapolated (RELEXT) methods reduce the number of iterations needed to reach residuals of 10^{-8}, 10^{-7} and 10^{-6} by 18%, 20% and 21%, respectively. By using $\alpha = 0.85$, for the webbase-2001 matrix, only between 62 and 89 Power iterations are needed for convergence giving a satisfactory approximation to the exact solution. However, for values of α close to 1 such as $\alpha = 0.99$, the Power method converges slowly needing $904, 1131$ or 1358 iterations, depending on the required tolerance (ϵ). In these cases, the RELEXT methods proposed here improve convergence considerably relative to the Power method by reducing the number of iterations needed to reach residuals of 10^{-8}, 10^{-7} and 10^{-6} by 35.4%, 40.6% and 45.8%, respectively. Note that the Simple Extrapolation ($r = 1$) is not effective (see Figure 1(a)) and slows down the convergence of the Power method. It is due to the fact that the Simple Extrapolation assumes that α is the only eigenvalue of modulus α and this is inaccurate; see e.g., [6]. On the other hand, the choice of r is very dependent of the condition number of the problem. Our experience indicates that, for our datasets, a good choice of the values of r for a well-conditioned problem ($\alpha = 0.85$) in the Extrapolated Power method is $r = 6$, while for values of α close to 1 (such as $\alpha = 0.99$) small values of r get poor results and it should be chosen greater than or equal to

$\alpha = 0.85$	$\epsilon = 10^{-8}$			$\epsilon = 10^{-7}$			$\epsilon = 10^{-6}$		
Matrix	REL	EXT	RELEXT	REL	EXT	RELEXT	REL	EXT	RELEXT
webbase-2001	6.7	15.7	18.0	7.5	17.5	20.0	8.1	17.7	21.0
it-2004	3.4	14.9	16.1	5.1	16.7	17.9	5.0	16.7	18.3
uk-2007-05	1.2	13.3	13.3	0.0	12.3	12.3	0.0	12.5	14.3
$\alpha = 0.99$	$\epsilon = 10^{-8}$			$\epsilon = 10^{-7}$			$\epsilon = 10^{-6}$		
Matrix	REL	EXT	RELEXT	REL	EXT	RELEXT	REL	EXT	RELEXT
webbase-2001	26.4	21.9	35.4	28.8	22.4	40.6	31.7	21.9	45.8
it-2004	21.6	19.0	31.2	23.4	19.8	36.2	25.8	21.2	41.2
uk-2007-05	10.2	14.0	19.0	11.0	14.8	21.7	12.1	17.2	27.2

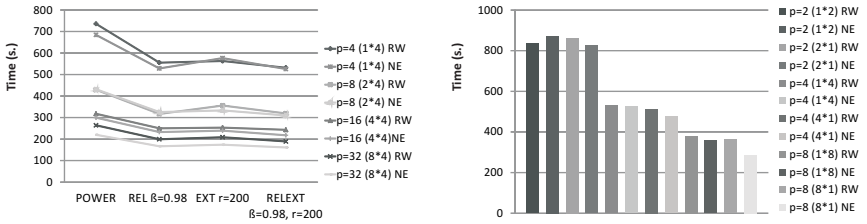
Table 2 Reduction in the number of iterations of the best Relaxed and/or Extrapolated methods with respect to the Power method (%).

50, obtaining the best results for r between 100 and 300. On the other hand, in both cases, good choices of β for the Relaxed Power method are between 0.97 and 0.99. Combining the relaxation and extrapolation such as is indicated in Algorithm 5, for $\alpha = 0.85$, the best results have been obtained for $r = 6$ and β between 0.97 and 0.99, and for $\alpha = 0.99$ the best results have been obtained for $r = 100$ and β between 0.98 and 0.99. Table 2 shows the reduction in the number of iterations achieved by the best Relaxed and/or Extrapolated methods for the three matrices of Table 1. The stopping criteria used in the rest of figures have been chosen such that $\epsilon = 10^{-8}$ for $\alpha = 0.85$ and $\epsilon = 10^{-6}$ for $\alpha = 0.99$. For $\alpha = 0.85$ and $\epsilon = 10^{-8}$ the RELEXT methods reduce the number of iterations relative to the Power method by 16.1% for the it-2004 matrix and by 13.3% for the uk-2007-05 matrix. While for $\alpha = 0.99$ and $\epsilon = 10^{-6}$ the RELEXT methods reduce the number of iterations by 41.2% for the it-2004 matrix and by 27.2% for the uk-2007-05 matrix. By analyzing the performance of the data distribution strategies used, we have obtained that, generally, the nonzero elements partitioning is the best distribution strategy for all the algorithms discussed herein, more specially as the number of processes increases. Figure 3 and Figure 4 illustrate this fact on shared memory (SM), distributed memory (DM), and distributed shared memory (DSM). Figure 5 and Figure 6(a) show the time that the described parallel algorithms take for computing PageRank using nonzero elements partitioning and varying the number of processes. The performance of these algorithms is affected by the fact that, at each iteration, the computational cost is not very high, while some communications among processes are required in order to proceed with the next iteration. Then, in order to deal with larger problems and to use all the available memory in a distributed system, the best strategy of parallelization needs to use at the same time the benefits of shared and distributed memory multiprocessors. Usually the best parallel results have been obtained using 1, 2 or 4 threads in each node. Moreover, it seems inappropriate to use more than 8 cores on a single node. Figure 6(b) shows the global time saving achieved for the it-2004 matrix. Figure 7(a) displays the running time per iteration of the RELEXT method for the uk-2007-05 matrix and the speed-up per iteration with respect to the Power method.



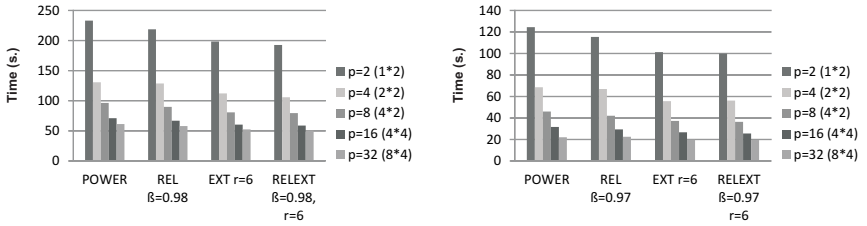
(a) SM $p = 1 * c$. (b) DM $p = s * 1$. (c) DSM $p = s * 4$.

Fig. 3 Row-wise versus nonzero elements distribution. Parallel Power method, $\alpha = 0.85$, webbase-2001.



(a) DSM $p = s * 4$. (b) SM $p = 1 * c$ and DM $p = s * 1$. RELEXT $r = 200, \beta = 0.98$

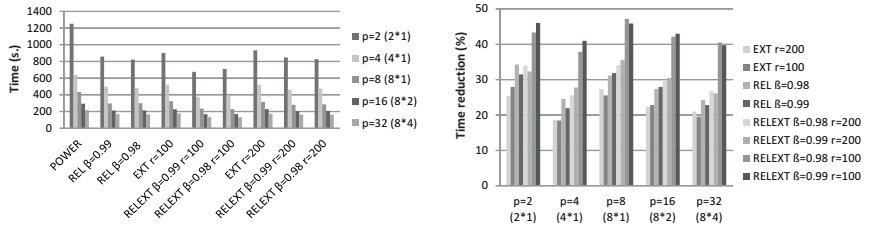
Fig. 4 Row-wise (RW) versus nonzero elements (NE) distribution. Parallel Relaxed and/or Extrapolated methods, $\alpha = 0.99$, it-2004.



(a) webbase-2001. (b) it-2004.

Fig. 5 Parallel Relaxed and/or Extrapolated methods, nonzero elements distribution, $\alpha = 0.85$.

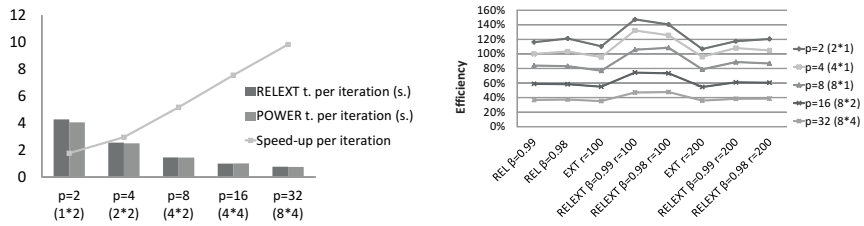
As it can be seen, reducing the number of iterations, using parallel Extrapolated Relaxed algorithms, comes at the expense of a slight increase in the work per iteration relative to the parallel Power algorithm. However, this overhead per iteration is minimal and then these proposed acceleration methods are beneficial. Concretely, for $\alpha = 0.85$ the parallel RELEXT algorithms accelerated the convergence saving between 15% and 20% in the time needed by the parallel Power method to reach a residual of 10^{-8}. For $\alpha = 0.99$, the time saving of these algorithms with respect to the parallel Power method to



(a) Total running times.

(b) Percentage of time reduction with respect to the parallel Power method.

Fig. 6 Parallel Relaxed and/or Extrapolated methods, nonzero elements distribution, $\alpha = 0.99$, it-2004.



(a) Times and speed-up per iteration, $\alpha = 0.85$, uk-2007-05.

(b) Efficiency, $\alpha = 0.99$, it-2004.

Fig. 7 Efficiency of the parallel Relaxed and/or Extrapolated methods, nonzero elements distribution.

	$\epsilon = 10^{-3}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-7}$
POWER It.	145	531	964
In/Out It. gain	13.8%	12.8%	11.1%
In/Out time gain	15.8%	15.3%	13.7%
RELEXT It. gain	18.6%	31.8%	21.7%
RELEXT time gain	22.2%	34.3%	26.4%

Table 3 RELEXT versus In/Out methods, $\alpha = 0.99$, SM $p = 1 * 8$, uk-2007-05.

reach a residual of 10^{-6} was more significant obtaining a time reduction between 40% and 50% for the webbase-2001 and it-2004 matrices and between 25% and 30% for the uk-2007-05 matrix. A considerable speed-up has been obtained (see Figure 7(b)), specially when α is close to 1. Concretely, the global efficiencies achieved in this case for the best RELEXT algorithms setting the sequential Power method as reference algorithm were about $140\% - 150\%$ for $p = 2$, $120\% - 130\%$ for $p = 4$, $105\% - 108\%$ for $p = 8$, $73\% - 75\%$ for $p = 16$, and $45\% - 50\%$ for $p = 32$. To conclude the analysis of the effectiveness of the parallel RELEXT algorithms, we have compared them with the inner-outer (In/Out) iterative algorithms proposed in [4]. Table 3 illustrates the gain obtained by both methods in relation to the Power method. As it can be seen,

our parallel RELEXT algorithms accelerate the PageRank computation more significantly than these inner-outer algorithms.

5 Conclusions

We have made an analysis of parallel algorithms based on the Power method and the use of relaxation and/or extrapolation techniques for accelerating the computation of PageRank. Two strategies of data distribution have been used: row-wise partitioning and nonzero elements partitioning. An hybrid implementation has been designed by combining various OpenMP threads for each MPI process. The results show that the best strategy of data distribution is the nonzero elements partitioning. Moreover, the proposed parallel Relaxed Extrapolated algorithms have a better convergence rate and can speed up the convergence time significantly with respect to the parallel Power algorithm.

References

1. Berkhin P (2005) A Survey on PageRank Computing. Internet Math 2(1):73–120
2. Boldi P, Codenotti B, Santini M, Vigna S (2004) Ubicrawler: A scalable fully distributed Web crawler. Softw Pract Expe 34:711–726
3. Dongarra J, Huss-Lederman S, Otto S, Snir M, Walkel D (1996) MPI: The complete reference. The MIT Press, Cambridge
4. Gleich D, Gray A, Greif C, Lau T (2010) An inner-outer iteration for computing PageRank. SIAM J Sci Comput 32(1):349–371
5. Gleich D, Zhukov L, Berkhin P (2005) Fast Parallel PageRank: A linear system approach. In The Fourteenth International World Wide Web Conference. ACM Press, New York
6. Kamvar SD (2010) Numerical Algorithms for Personalized Search in Self-organizing Information Networks. Princeton University Press, New Jersey
7. Kamvar SD, Haveliwala TH, Golub GH (2004) Adaptive Methods for the Computation of PageRank. Linear Algebra Appl 386:51–65
8. Kamvar SD, Haveliwala TH, Manning CD, Golub GH (2003) Exploiting the Block Structure of the Web for Computing PageRank. Stanford University Technical Report, SCCM-03-02
9. Kamvar SD, Haveliwala TH, Manning CD, Golub GH (2003) Extrapolation Methods for Accelerating PageRank Computations. In Twelfth International World Wide Web Conference 261–270
10. Migallón H, Migallón V, Palomino JA, Penadés J (2010) Parallelization Strategies for Computing PageRank. In Proceedings of the Seventh International Conference on Engineering Computational Technology, Civil-Comp Press, Stirlingshire, UK, Paper 29, doi:10.4203/ccp.94.29
11. OpenMP official site (2008) openmp.org
12. Page L, Brin S, Motwani R, Winograd T (1999) The PageRank citation ranking: Bringing order to the Web. Technical Report, Stanford Digital Library Technologies Project
13. Rungsawang A, Manaskasemsak B (2012) Fast PageRank Computation on a GPU Cluster. In 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing 450–456
14. Rungsawang A, Manaskasemsak B (2006) Parallel adaptive technique for computing PageRank. In Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP'06 15–20
15. Wilkinson JH (1998) The algebraic eigenvalue problem. Oxford University Press, Oxford
16. Wu G, Wei Y (2010) An Arnoldi-Extrapolation algorithm for computing PageRank. J Comput Appl Math 234:3196–3212
17. Zhang H, Goel A, Govindan R, Mason K, Van Roy B (2004) Making Eigenvector-Based Reputation Systems Robust to Collusion, Lect Notes Comput Sc 3243:92–104